Conceptual Navigation in RDF Graphs with SPARQL-like Queries

Sébastien Ferré

IRISA, Université de Rennes 1
Campus de Beaulieu, 35042 Rennes cedex, France
Email: ferre@irisa.fr

Abstract Concept lattices have been successfully used for information retrieval and browsing. They offer the advantage of combining querying and navigation in a consistent way. *Conceptual navigation* is more flexible than hierarchical navigation, and easier to use than plain querying. It has already been applied to formal, logical, and relational contexts, but its application to the semantic web is a challenge because of inference mechanisms and expressive query languages such as SPARQL. The contribution of this paper is to extend conceptual navigation to the browsing of RDF graphs, where concepts are accessed through SPARQL-like queries. This extended conceptual navigation is proved *consistent* w.r.t. the context (i.e., never leads to an empty result set), and *complete* w.r.t. the conjunctive fragment of the query language (i.e., every query can be reached by navigation only). Our query language has an *expressivity* similar to SPARQL, and has a more *natural* syntax close to description logics.

1 Introduction

With the growing amount of available resources in the Semantic Web (SW), it is a key issue to provide an easy and effective access to them, not only to specialists, but also to casual users. The challenge is not only to allow users to retrieve particular resources (e.g., flights), but to support them in the exploration of a domain knowledge (e.g., which are the destinations? Which are the most frequent? With which companies and at which price?). We call the first mode *retrieval search*, and, following Marchionini [Mar06], the second mode *exploratory search*. The latter is generally supported by *faceted search* [ST09].

Conceptual navigation, based on Formal Concept Analysis (FCA) [GW99], also supports exploratory search by guiding users from concept to concept [CR96,DE08,Fer09]. The concept lattice plays the role of an exploration space, where each concept can be reached either by entering a query, or by following navigation links between concepts. At each step of the navigation, the set of navigation links is organized as a summary of the extent of the current concept, and provides insight and feedback about the context, and thus supports exploratory search. This solves the dilemma between using an expressive query

language that is difficult to use (e.g., Boolean queries), and an intuitive but rigid navigation structure (e.g., file hierarchies).

Languages of the SW, on the one hand, are more expressive than FCA, w.r.t. both the representation language (e.g., RDFS vs formal context) and the query language (e.g., SPARQL vs sets of attributes). Extensions of FCA such as Logical Concept Analysis (LCA) with relations [FRS05] or Relational Concept Analysis (RCA) [HHNV07] get closer to SW languages but each extension still misses large fragments of expressivity: e.g., LCA misses cycles in queries, RCA misses disjunction. On the other hand, querying languages for the SW (e.g., SPARQL [PAG06], OWL-QL [FHH04]), while expressive, are difficult to use, even for specialists, and do not provide enough feedback to satisfy exploratory search. Indeed, even if users have a perfect knowledge of the syntax and semantics of the query language, they may be ignorant about the application vocabulary, i.e., the *ontology*. If they also master the ontology or if they use a query assistant (e.g., Protégé[1]), the query will be syntactically correct and semantically consistent w.r.t. the ontology but can still produce no result (e.g., it makes sense to ask for a flight from Rennes to Agadir, but it happens there is none). Faceted search systems such as Slashfacet [HvOH06] or BrowseRDF [ODD06] rely on actual data instead of an ontology to assist users in their search. They do support exploratory search but with limited expressivity compared to SW query languages. For instance, they allow neither for cycles in queries, nor for general disjunction and negation.

The contribution of this paper is to adapt and extend conceptual navigation to the Semantic Web. We propose a navigation process that (1) is based on a query language whose *expressivity* is similar to SPARQL, and (2) has a *natural* and concise notation (similar to N3[2]), and that is (3) *consistent* (no dead-end) and (4) *complete* (every query can be reached by navigation). The last two points give a formal basis to conceptual navigation, and make it a real alternative, rather than a complement, to querying. Our approach builds upon, and is compatible with, existing techniques for designing and storing ontologies, reasoning, as well as querying languages and their implementations.

We first give basics of the Semantic Web (Section 2), and Logical Information Systems (Section 3) from which our extension starts. We then detail our proposal for conceptual navigation in the Semantic Web, separating the static part (user interface as a local view on the concept lattice, Section 4), and the dynamic part (user interactions as navigation links between concepts, Section 5). A few perspectives are discussed before concluding (Section 6).

2 Basics of the Semantic Web

The Semantic Web is founded on several representation languages, such as RDF, RDFS, and OWL, which provide increasing inference capabilities [HKR09]. The two basic units of these languages are *resources* and *triples*. A resource can

[1] See `http://protege.stanford.edu/`

[2] See `http://www.w3.org/DesignIssues/Notation3.html`

be either a URI (Uniform Resource Identifier), a literal (e.g., a string, a number, a date), or a *blank node*, i.e., an anonymous resource. A URI is the absolute name of a *resource*, i.e., an entity, and plays the same role as URL w.r.t. web pages. Like URLs, a URI can be a long and cumbersome string (e.g., `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`), so that it is often denoted by a qualified name (e.g., `rdf:type`). A triple (s, p, o) is made of 3 resources, and can be read as a simple sentence, where s is the subject, p is the verb (called the predicate), and o is the object. For instance, the triple `(ex:Bob,rdf:type,ex:man)` says that "Bob has type man", or simply "Bob is a man". Here, the resource `ex:man` is used as a class, and `rdf:type` is used as a property, i.e., a binary relation. The triple `(ex:Bob,ex:friend,ex:Alice)` says that "Bob has friend Alice", where `ex:friend` is another property. The triple `(ex:man,rdfs:subClassOf,ex:person)` says that "man is subsumed by person", or simply "every man is a person". The set of all triples of a knowledge base form a RDF graph. A RDF graph that uses the OWL vocabulary to define classes and properties is generally called an *ontology*.

RDF(S) introduces a vocabulary of resources to represent the membership to a class (`rdf:type`), subsumption between classes (`rdfs:subClassOf`) and between properties (`rdfs:subPropertyOf`), the domain (`rdfs:domain`) and range (`rdfs:range`) of properties, the meta-classes of classes (`rdfs:Class`) and of properties (`rdf:Property`), etc. OWL introduces additional vocabulary to represent complex classes and properties: e.g., restrictions on properties, intersection of classes, inverse property. The variant OWL-DL is the counterpart of Description Logics (DL) [BCM+03], where resources are individuals, classes are concepts, and properties are roles. Each language comes with a semantics, and the richer the vocabulary is, the more expressive and the more complex the inference is. In this paper, we do not make any strong assumption on the vocabulary.

Query languages provide on SW knowledge bases the same service as SQL on relational databases. They generally assume that implicit triples have been inferred and added to the base. The most well-known query language, SPARQL [PAG06], reuse the `SELECT FROM WHERE` shape of SQL queries, using graph patterns in the WHERE clause. For instance, twin siblings can be retrieved by the following query:

```
SELECT ?x ?y FROM <ex.rdf> WHERE { { ?x ex:mother ?z. ?y ex.mother
 ?z. ?x ex:birthdate ?d. ?y ex:birthdate ?d } FILTER (?x != ?y) }
```

Two persons `?x` and `?y` are twins if they share a same mother and a same birthdate, and are different. The FILTER condition is necessary because nothing prevents two variables to be bound to a same resource.

There exists a mapping from RCA to DL [HHNV07], or equivalently from RCA to OWL-DL. In short, it maps objects to resources, attributes to classes, the incidence relation to the property `rdf:type`, each context of a Relational Context Family (RCF) to a class, and each relation of the RCF to a property. Formal concepts are mapped to defined OWL classes, and subconcept links are mapped to the property `rdfs:subClassOf`. This suggests that the SW standard formats based on XML can be used to represent and share FCA data, both formal

contexts and formal concept lattices. Therefore, every algorithm or system that works on SW data does work on FCA data. Conversely, a challenge for FCA is to stretch its algorithms and systems so that they work on SW data [RKH07]. Previous work have mostly focused on the use of FCA to support the design of ontologies: e.g., implication basis for description logics [BD08], acquisition of OWL axioms based on attributed exploration [VR08]. The contribution of this paper is the extension of FCA-based conceptual navigation to the semantic web. We take Logical Information Systems (LIS) [Fer09] as a starting point because they share, at a lower level, expressive query languages and inference.

3 Basics of Logical Information Systems

We here recall the basics of Logical Information Systems (LIS) because we start from the structure of its user interface and interactions to conceptual navigation on RDF graphs. LIS instantiate both the conceptual navigation paradigm [Fer09], and the faceted search paradigm [ST09][3]. LIS user interface gives a *local view* of the concept lattice, centered on a concept called the *focus*. The local view is made of three parts: (1) the query, (2) the extent, and (3) the index. The query is a logical formula. The extent is the set of objects that are matched by the query, along the principles of logical concept analysis [FR04]. The extent identifies the focus concept. Finally, the index is a finite subset of the logic that is restricted to formulas that match at least one object in the extent. The index plays the role of a summary or inventory of the extent, showing which kinds of objects there are, and how many of each kind there are.

The query can be modified in three ways. To *query by formula* is to directly edit the query, which requires expertise or luck from the user. To *navigate* is to select formulas in the index in order to make the query more specific (moving downward in the lattice) or more general (moving upward in the lattice). To *query by examples* is to select a set of objects in the extent, which leads to the concept whose intent (the new query) is the conjunction of all properties shared by the selected objects. In the three cases, the modification of the query entails the update of the extent, hence updating the focus concept and the index. By definition of the index, no navigation link (a selection in the index) can lead to an empty result. Conversely, because the navigation structure is a lattice rather than a hierarchy, all valid conjunctions of formulas can be reached by navigation, and in any order. Contrary to querying by formula, navigation only requires from the user to recognize the meaning of formulas in the index, in the context of the application.

4 User Interface: Local View

The extension of the LIS framework to the semantic web applies to the query language and navigation modes, highly improving expressivity and flexibility

[3] See Chapters 3, 4, 5, 8, and 9 of this book.

when browsing a dataset. In this section, we redefine the LIS notion of *local view*. This comprises the definition of queries and their extents, and the summarization index over extents. Navigation links are defined in Section 5 on top of the local view.

We consider the dataset to be a RDF graph [PAG06], which may include both explicit and implicit facts (i.e., triples) through reasoning. In this paper, we do not make the distinction between the two. In practice, the implicit facts may be materialized in a preprocessing stage, or infered on demand. We assume pairwise disjoint infinite sets of URIs (U), blank nodes (B), and literals (L). The set of *resources* is defined as $R = U \cup B \cup L$.

Definition 1 (RDF graph). *An* RDF graph *is defined as a set of* triples $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$*, where* s *is the* subject*,* p *is the* predicate*, and* o *is the* object*.*

RDF vocabulary for genealogy. For illustration purposes, we consider RDF graphs about genealogical data. The URIs of this domain are associated to a namespace `gen:`. This prefix is omitted if there is no ambiguity. Resources can be *persons*, *events*, *places* or literals such as names or dates. Persons belong either to the class of *men* or to the class of *women*, may have a *firstname*, a *lastname*, a *sex*, a *father*, a *mother*, a *spouse*, a *birth*, and a *death*. A birth or a death is an event that may have a *date* and a *place*. Places can be described as *parts* of larger places. OWL axioms may be used to enforce some invariants, e.g., the property *spouse* is symmetrical, the property *father* is functional, and the property *part* is transitive.

Figure 1 shows the user interface of our prototype, applied to the genealogy of George Washington[4]. It reflects the structure of a local view with the query at the top, the extent at the left, and the index at the center and right. The query selects "male persons whose lastname is Washington". There are 17 answers in the extent: e.g., George Washington. The central index shows that 7 of them have a known birth's place, and that 11 of them are known to be married. The right index shows their distribution in a taxonomy of locations, according to their birth's place. The hidden tabs give their distribution according to their birth's year or firstname.

4.1 Queries and Extensions

A SPARQL query can be used to define the focus of a local view. For instance, a local view that puts the focus on the set of "women whose some parent was born in Virginia in 1642" can be defined by the following SPARQL query.

```
SELECT ?x
WHERE { ?x rdf:type gen:woman. ?x gen:parent ?p. ?p gen:birth ?b.
        ?b gen:date 1642. ?b gen:place ?l. gen:VA gen:part ?l. }
```

[4] Downloadable at `http://www.irisa.fr/LIS/ferre/camelis/camelis2.html`

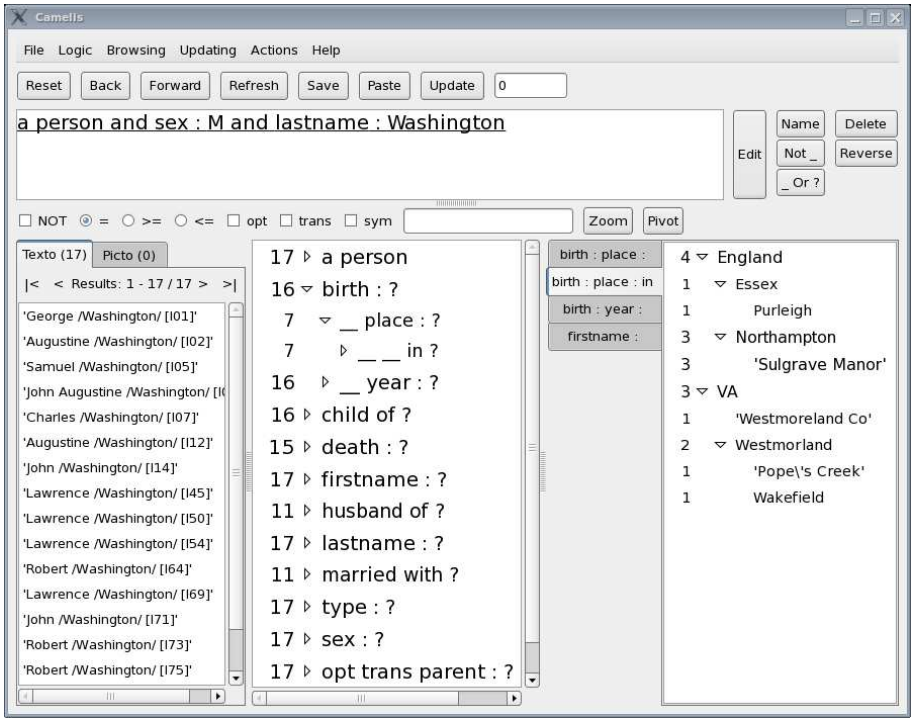

Figure1. A local view: query (top), extension (left), and index (center and right). The query selects male persons whose lastname is Washington.

In LIS, a query must necessarily define a set of resources, i.e., a mono-dimensional relation. This implies that we only need SPARQL queries with a single variable in the SELECT clause. This also means that our queries are analogous to OWL complex classes. In fact, the above query can be expressed in the description logic $\mathcal{SHOIN}$ that backs OWL-DL:

$$Woman \sqcap \exists parent.\exists birth.(\exists date.\{1642\} \sqcap \exists place.\exists part^{-}.\{VA\}).$$

The advantages of the DL syntax is that it is more concise, and that it avoids the use of variables. LIS do not require from end-users the ability to write queries, but they do require from them to understand queries. Ideally, the queries should be understandable with little, if any, learning. We think that the DL syntax is closer to this objective than the SPARQL syntax, provided that mathematical symbols are replaced by words, of course. At the same time, SPARQL has more expressive patterns (e.g., cycles).

We propose a new language for querying RDF graphs, where query results are sets of resources. Therefore, the expressions of this language are called *complex classes*, and make use of *complex properties*, derived from basic properties.

Definition 2 (complex property). *A* complex property *is any of:*

p `:`	*the property p itself,*
p `of`	*the inverse of the property p,*
p `with`	*the symmetric closure of the property p,*
`trans` P	*the transitive closure of the complex property P ("transitively P"),*
`opt` P	*the reflexive closure of the complex property P ("optionally P").*

Applying the three closures, `opt trans` p `with`, defines an equivalence relation, while `opt trans` P defines a partial ordering if P is antisymmetric, and a pre-order otherwise. In the following, we use `in` as an abbreviation for the complex property `opt trans part of`.

Definition 3 (complex class). *Let V be an infinite set of variables, disjoint with the set of resources R. For every resource $r \in R$, variable $v \in V$, URI $u \in U$, complex property P, and complex classes C, C_1, C_2, the following expressions are also complex classes (in decreasing priority for operators):*

$$r \mid \texttt{?}v \mid \texttt{?} \mid \texttt{a}\ u \mid P\ C \mid \mathit{not}\ C_1 \mid C_1\ \texttt{and}\ C_2 \mid C_1\ \texttt{or}\ C_2.$$

Compared to DL languages, the complex class r corresponds to the *nominal* $\{r\}$, the anonymous variable `?` corresponds to the *top concept* $\top$, the expression $P\ C$ corresponds to a *qualified existential restriction* $\exists P.C$ (simply a *restriction* from now on), the expression `a` u corresponds to a *concept name*, the `and` corresponds to *concept intersection* $\sqcap$, the `or` corresponds to *concept union* $\sqcup$, and `not` corresponds to *concept complement* $\neg$. The addition of variables `?`v allows for the expression of cyclical graph patterns, like in SPARQL. The notation p `:` is reminiscent of the notation of valued attributes. For example, in the expression `name : "John"`, `name` is the attribute, and `"John"` is the value. The expression can be read "has name John", or "whose name is John". The above query can now be written:

```
a woman and parent : birth : (date : 1642 and place : in VA)
```

A semantics for our language, and a practical way to compute answers to queries in this language, is obtained by defining a translation to one-dimensional SPARQL queries. Graph patterns are given in the abstract syntax (constructs: AND, UNION, FILTER) defined in [PAG06], rather than the (equivalent) concrete SPARQL syntax, for the sake of simplicity and because it provides a necessary extension for translating negation (construct: MINUS). The empty graph pattern is denoted by 1.

Definition 4. *Let C be a complex class. The SPARQL translation of C is defined by*

$$\Gamma(C) = \texttt{SELECT ?}x\ \texttt{WHERE}\ f(g)$$

where $x \in V$ is a fresh variable not occurring in C, and $(g, f) = \gamma(x, C)$ (f is a function). The table below defines γ by induction on complex classes and complex properties. $\gamma(x, C)$ returns a graph pattern g, and a graph pattern modifier f,

that together represent the fact that x is an instance of the complex class C. $\gamma(x, P^\alpha, y)$ is a graph pattern representing the complex property P between x and y, under the relation closure α. α is a subset of $\{?, +\}$, where ? (resp. +) denotes the reflexive (resp. transitive) closure of a binary relation. For every $i \in \mathbb{N}$, we assume $(g_i, f_i) = \gamma(x, C_i)$.

expression	graph pattern	graph pattern modifier
r	1	$\lambda g.(g$ `FILTER ?`$x = r)$
`?`v	1	$\lambda g.(g$ `FILTER ?`$x = ?v)$
`?`	1	$\lambda g.g$
`a` u	$(?x, \texttt{rdf:type}, u)$	$\lambda g.g$
$P\ C$	g_1 `AND` g_2	f_2
	where y is a fresh variable, $g_1 = \gamma(x, P^\emptyset, y)$, $(g_2, f_2) = \gamma(y, C)$	
C_1 `and` C_2	g_1 `AND` g_2	$\lambda g.(f_2(f_1(g)))$
`not` C_1	1	$\lambda g.(g$ `MINUS` $f_1(g_1))$
C_1 `or` C_2	1	$\lambda g.(g$ `AND` $(f_1(g_1)$ `UNION` $f_2(g_2)))$
p `:`	$(?x, p^\alpha, ?y)$	
p `of`	$(?y, p^\alpha, ?x)$	
p `with`	$(?x, p^\alpha, ?y)$ `UNION` $(?y, p^\alpha, ?x)$	
`opt` P	$\gamma(x, P^{\{?\}\cup\alpha}, y)$	
`trans` P	$\gamma(x, P^{\{+\}\cup\alpha}, y)$	

Compared to SPARQL, our language is restricted to one-dimensional relations, which makes the SPARQL contruct OPT irrelevant. This restriction is balanced to some extent by navigation (see end of Section 4.2). SPARQL allows for variables in predicate position, which is not directly possible in our language, but indirecly possible through the reification of triples. However, our language has native general negation, and reflexive/transitive closure. Perez et al. have shown that general negation is expressible in SPARQL, but in a very cumbersome way [PAG06].

We can now define the second part of a local view, the extent. It is simply defined as the answers to the SPARQL translation of the query.

Definition 5 (extent). *Let C be a complex class. The* extent *of C, noted $ext(C)$, is the set of resources that are answers to its SPARQL translation $\Gamma(C)$. Every element of the extent is called an* instance *of the complex class C.*

The extent of a query determines the focus concept the query leads to. The definition and the computation of the intent of this concept is not necessary in our framework. The intensional part of the local view is played by the *index*.

4.2 Summarization Index

The third part of the local view is the *index* which serves as a summary of the extent. Every *index term* is a descriptor of some or all resources in the extent. Therefore, every index term can be seen either as part of the intent of the focus

concept, when shared by *all* instances; or as a refinement of the query, when shared by *some* of the instances.

Definition 6 (index term and intent term). *Let q be a complex class representing the query of a local view, and C be a complex class that contains only variables that also occur in q. C is an* index term *of q, which we note $C \in index(q)$, if $ext(q \texttt{ and } C) \neq \emptyset$. C is an* intent term *of q, which we note $C \in int(q)$, if $ext(q \texttt{ and not } C) = \emptyset$.*

The number of index terms can be infinite, but in practice only a limited subset is presented to the user at any given time. Initially, a small index is presented, and then the user can expand it in a controlled way to see more index terms.

Instead of presenting the index terms as a flat list, they can be organized into a partial ordering $\leq$ that reflects subsumption relationships between them. Figure 1 shows how this partial ordering can be rendered as trees of complex classes. The number at the left of each index term is its count. Figure 1 gives on the right side the number of male Washington born in each place. This partial ordering needs not be complete w.r.t. subsumption because it does not affect query answering. The guiding criteria to design this partial ordering is that it should: be intuitive to users (i.e., they can anticipate the infered subsumptions), provide enough structure to the index, and be of practical complexity.

In the illustrations of this paper, we use RDFS inference through the properties `rdfs:subClassOf` and `rdfs:subPropertyOf`. In the partially ordered index, every class is placed under its superclasses. For instance, `a woman` $\leq$ `a person` $\leq$ `a thing`. Every property is placed under its superproperties, and also under its different closures. For instance, `father of` $\leq$ `parent of` $\leq$ `trans parent of`. From this ordering of complex properties, restrictions can also be ordered: $P_1\ C_1 \leq P_2\ C_2 \iff P_1 \leq P_2 \ \wedge\ C_1 \leq C_2$. For instance, `father of a man` $\leq$ `parent of a person`. Every index term is placed under the anonymous variable `?`, which then plays the role of the root of the whole index. Similarly, all restrictions in the form $P\ C$ are grouped under P `?`.

We have noted above that a reflexive and transitive complex property, i.e., in the form of `opt trans` P, is a partial ordering. This partial ordering can be used to organize the index because the following subsumption holds for every complex property P, and every resources r_1, r_2: `opt trans` $P\, r_1 \leq$ `opt trans` $P\, r_2 \iff r_1 \in ext($`opt trans` $P\ r_2)$. This is illustrated in Figure 1, on the right side, by the birthplace of male Washington's (recall that `in` = `opt trans part of`): e.g., `birth : place : in Westmorland` $\leq$ `birth : place : in VA`, because `Westmorland` $\in ext($`in VA`$)$ ("Westmorland is in VA"). Therefore, the tree under the index term `birth : place : in ?` forms a taxonomy of locations, even if each location is represented in the RDF graph as a resource, and not as a class. Similarly, a descendancy chart of the ancestors of the selected people is obtained under the index term `opt trans parent : ?`, showing under each individual its children, and this recursively.

The index alleviates to some extent our restriction to one-dimensional queries. Assume the SPARQL query `SELECT ?x ?y WHERE { ?x rdf:type`

`gen:man . ?x gen:mother ?y }`. By setting the query to `a man`, and by expanding the index term `mother : ?`, the index gives the list of mothers of a man, and for each mother, how many male children she has. A highlighting mechanism allows to select a man in the extent to discover who is his mother; and symmetrically, to select a mother in the index to discover which are her children. This presentation is also more compact than a listing of all pairs (man, mother).

5 User Interaction: Navigation Links

A local view is determined by its query. The query determines the extent and the index as presented above. By default, the user is initially presented with the local view of the most general query `?`, whose extent is the set of all resources defined in the dataset. In their search for information, users need to change the focus, i.e., to change the current query. In retrieval search, users are looking for a particular set of resources, and try and find the query whose extent matches this set of resources. For instance, to answer the question "Which women are married to a Washington?", we can use the query `a woman and married with lastname : Washington`. In exploratory search, users are looking for patterns in data rather than for particular resources. For instance, if users is interested in the birthplace of people having lastname "Washington", they can set the query to `lastname : Washington`, and then explore the index under the index term `birth : place : in ?`. They obtain a natural hierarchy of places, where each place is annotated by the proportion of the Washington's that were born in this place. The index also informs about their birthdates, ancestors, descendants, etc.

A well informed user can of course directly type in queries. However, as explained in the introduction, this requires not only to have good knowledge of the query language (syntax and semantics) and of the domain-specific vocabulary (e.g., `man`, `place`), but also of the contents of the dataset if one wants to avoid empty results. This is paradoxical as the less we know a dataset, the more we need to search in it. We propose to define *navigation graphs* whose nodes are queries, hence local views, and whose edges are *navigation links* that users can follow.

Definition 7 (navigation graph). *A* navigation link *is a triple* $(q\ l\ q')$*, where* q *is the* source query*,* q' *is the* target query*, and* l *is the* label *of the navigation link. A* navigation graph G *is a set of navigation links that is deterministic, i.e., if* $(q\ l\ q'_1)$ *and* $(q\ l\ q'_2)$ *are 2 navigation links of* G*, then* $q'_1 \equiv q'_2$ *(*$\equiv$ *denotes query equivalence).*

Definition 8 (local links). *Let* G *be a navigation graph, and* q *be a query. The* local links *of* q *in* G*, noted* $Links_G(q)$*, is the set of navigation links in* G *whose source query is* q*. A navigation graph* G *is* locally finite *iff* $Links_G(q)$ *is finite for every complex class* q*.*

In the following, we first define the different kinds of navigation links, i.e., the *navigation modes*. Then a navigation graph is proved consistent, i.e., never leads

to empty results. Finally, a locally finite navigation graph is proved complete, e.g., can lead to arbitrary queries/local views. These two navigation graphs define bounds between which every navigation graph is both consistent and complete.

5.1 Navigation Modes

There are only three navigation modes: *zoom-in*, *naming*, and *reversal*. Each navigation mode determines the target query in function of the source query and an additional argument. A zoom-in applies to a complex class, a naming applies to a variable name (generated or user-given), and a reversal applies to a part of the source query. If we see the query in its SPARQL form, the zoom-in extends the graph pattern, while the reversal changes the variable in the SELECT clause. The naming makes a variable of the SPARQL query visible in the LIS query.

Definition 9 (zoom-in). *Let q be a query, and C be a complex class. A triple* `(`q `[zoom-in` C`]` q'`)` *is a* zoom-in *navigation link iff* $q' =$ `(`q `and` C`)`.

Zoom-in is mostly useful when the extent of the resulting query is strictly smaller and not empty. This is obtained when using index terms that are *not* intent terms. This useful distinction can be made visible in the interface by different renderings (e.g., font-color), and annotations (e.g., count).

Naming works similarly to zoom-in, but applies to a fresh variable, i.e., not occurring in the initial query, while zoom-in is expected to apply to variables already occurring in the source query.

Definition 10 (naming). *Let q be a query, and* `?`v *be a variable. A triple* `(`q `[naming ?`v`]` q'`)` *is a* naming *navigation link iff the variable v does not occur in q, and* $q' =$ `(`q `and ?`v`)`.

Naming does not change the extent, because it produces a query that is equivalent to the initial query, but it introduces a new variable in the query, and hence in the index. Subsequent zoom-in navigation links on these variables allow to form cycles in the graph pattern of the query.

Reversal does not change the graph pattern, but it changes the variable that appear in the SELECT clause. Indeed, in a LIS query, the focus is only on one variable, and it is useful to change this focus. Therefore, a reversal changes the extent, and hence the focus. A difficulty is that not all variables in the SPARQL pattern appear in its corresponding LIS query. In a reversal navigation link, the new variable is implicitly designated by a part of the query: i.e., an occurrence of a complex class or complex property in the query. Reversal is undefined when the part of the query is in the scope of union or complement.

Definition 11 (reversal). *Let q be a query, and e be a part of q. A triple* `(`q `[reversal` e`]` q'`)` *is a* reversal *navigation link iff e does not occur in the scope of a* union *or* complement*, and* $q' = \rho(\texttt{?}, \underline{q})$*. The following table defines $\rho(q', \underline{C})$ by induction on the complex class C. The underlined part indicates in which part of the query the selected element e stands. The first parameter q' is used as an accumulator in the building of the target query.*

complex class C	*result of* $\rho(q', \underline{C})$
$\underline{P}\ C'$ *when* $\sigma(P) = subject$	$\rho(q', \underline{P\ C'})$
$\underline{P}\ C'$ *when* $\sigma(P) = object$	$\rho(q', P\ \underline{C'})$
$P\ \underline{C'}$	$\rho(P^{-1}\ q', C')$
$\underline{C_1}$ `and` C_2	$\rho(q'$ `and` $C_2, C_1)$
C_1 `and` $\underline{C_2}$	$\rho(q'$ `and` $C_1, C_2)$
otherwise	q' `and` C

This definition needs a definition for the inverse of a complex property (P^{-1}), and for what a complex property refers to, whether the subject or the object of the property ($\sigma(P)$). The following table provides these definitions by induction on complex properties:

complex property P	*inverse* P^{-1}	*reference* ($\sigma(P)$)
p `:`	p `of`	*object*
p `of`	p `:`	*subject*
p `with`	p `with`	*subject*
`opt` P	`opt` P^{-1}	$\sigma(P)$
`trans` P	`trans` P^{-1}	$\sigma(P)$

We illustrate reversal with two examples, starting with the query already presented earlier: q = `a woman and parent : birth : (date : 1642 and place : in VA)`.

- q `[reversal place :] place of (birth of parent of a woman and date : 1642) and in VA`
  new focus on "where in Virginia a parent of a woman was born in 1642"
- q `[reversal 1642] 1642 and date of (place : in VA and birth of parent of a woman)`
  new focus on "the date 1642 of the birth in VA of a parent of a woman"

We now define a generic navigation graph, parameterized by a *vocabulary* of complex classes.

Definition 12 ($\mathcal{C}$-navigation graph). *Let $\mathcal{C}$ be a set of complex classes, called* vocabulary. *The $\mathcal{C}$-navigation graph G defines for every source query q the set of local navigation links, $Links_G(q)$, as follows:*
- a zoom-in link for each $C \in index(q) \cap \mathcal{C}$;
- a zoom-in link for each `not` C *s.t. $C \in \mathcal{C} \setminus int(q)$;*
- a naming link for one fresh variable;
- and a reversal link for each part of q not occurring in a union or complement.

5.2 Navigation Consistency

We first define *consistency* for navigation links and navigation graphs.

Definition 13 (navigation consistency). *A navigation link* `(`$q\ l\ q'$`)` *is* consistent *w.r.t. a dataset iff it preserves the existence of answers, i.e., $ext(q) \neq \emptyset$ implies $ext(q') \neq \emptyset$. A navigation graph is consistent iff its links are all consistent.*

A zoom-in link is consistent if it applies to an index term or to the complement of non-intent term of the query, so that every index term represents one or two consistent navigation links. All naming and reversal links are consistent.

Lemma 1. *Let q be a query. For every complex class $C \in index(q)$, the navigation link* `(`q `[zoom-in` C`]` q'`)` *is consistent; and for every complex class $C \notin int(q)$, the navigation link* `(`q `[zoom-in not` C`]` q'`)` *is consistent.*

Proof. By definition of $index(q)$, $int(q)$, and navigation link consistency. □

Lemma 2. *Let q be a query, and v be a variable not occurring in q. The navigation link* `(`q `[naming ?`v`]` q'`)` *is consistent.*

Proof. As `?`v does not occur in q, $q' = q$ `and ?`v is equivalent to q `and ?`, which is equivalent to q. □

Lemma 3. *Let q be a query, and e be a part of the query q not occurring in the scope of a union or complement. The navigation link* `(`q `[reversal` e`]` q'`)` *is consistent.*

Proof. It can be proved that the source and target query of a reversal link define the same SPARQL query, up to the renaming of variables, and the possible replacement of the variable in the SELECT clause. If the source query q has answers, this implies that every variable in the conjunctive part of the graph pattern (not in the scope of an union or complement pattern) has substitution values. Therefore, the new variable in the SELECT clause has substitution values. Hence, the target query q' also has answers. □

Theorem 1. *For every vocabulary $\mathcal{C}$, the $\mathcal{C}$-navigation graph is consistent.*

Proof. This is a direct consequence of previous definitions and lemmas. □

This implies that, given that the initial query has answers, a user who only follows navigation links in the navigation graph will never fall in a dead-end, which is a frequent cause of frustration in information systems.

5.3 Navigation Completeness

A navigation graph is complete if every query is reachable from the query `?`.

Definition 14 (navigation completeness). *A navigation graph G is* complete *iff for every query q whose extent is not empty, there exists a finite sequence of navigation links* `(`q_0 l_1 q_1 `...`q_{n-1} l_n q_n`)`*, where q_0 = `?`, $q_n = q$, and for every $i \in [1, n]$, $(q_{i-1}\ l_i\ q_i)$ is a navigation link of G. We call such a sequence, a* navigation path.

For practical use, a navigation graph has to be locally finite, i.e., only a finite set of local navigation links are suggested in any local view. Under this constraint we define a vocabulary for which completeness is achieved for *conjunctive queries*, i.e., queries without unions and with complements restricted to terms in $\mathcal{C}$.

A number of semantic web query languages are equivalent to conjunctive queries, and provide neither negation nor disjunction (e.g., OWL-QL [FHH04]). In our approach, negation and disjunction can still be introduced at any step of a navigation by editing the query by hand.

Theorem 2 (locally-finite conjunctive-complete navigation graph). *Let $\mathcal{C}$ be a finite vocabulary of complex classes containing at least resources (URIs, literals), variables, and unqualified restrictions P `?`. The $\mathcal{C}$-navigation graph G is locally-finite and complete for* conjunctive queries.

Proof. For every query q, the set of local navigation links $Links_G(q)$ is finite because in a given dataset (1) there is a finite number of URIs, and hence of properties, (2) only literals present in the extent belong to the index, and this extent is always finite, and (3) only variables occurring in q belong to the index.

For completeness, it suffices to prove that for every complex classes q_0, and C such that C is a conjunctive query and $ext(q_0$ `and` $C) \neq \emptyset$, there exists a path in G from q_0 to $q_1 = q_0$ `and` C. In particular, setting $q_0 =$ `?` and $C = q$, we obtain that there exists a path from `?` to `(? and` q`)`, which is equivalent to q. We proceed by induction on the complex class C:

$C = r$`:` there is a path `(`q_0 `[zoom-in` r`]` q_1`)` (because $r \in \mathcal{C}$)
$C =?v,\ ?v \in q_0$`:` there is a path `(`q_0 `[zoom-in` $?v$`]` q_1`)` (because $?v \in \mathcal{C}$)
$C =?v,\ ?v \notin q_0$`:` there is a path `(`q_0 `[naming` $?v$`]` q_1`)`
$C =?$`:` $q_1 = q_0$ `and ?` $\equiv q_0$
$C =$ `a` u`:` $C \equiv$ `rdf:type` u (Definition 4)
$C = P\ C'$`:` (1) there is a path `(`q_0 `[zoom-in` P `?]` q_0 `and` P `? [reversal ?]` P^{-1} q_0 `and ?` $\equiv$ P^{-1} q_0`)`, (2) there is path from P^{-1} q_0 to `(`P^{-1} q_0 `and` C'`)` by induction on C', (3) there is a path `(`P^{-1} q_0 `and` C' `[reversal` q_0`]` q_0 `and` $P\ C = q_1$`)`. Induction in (2) is justified because `(`P^{-1} q_0 `and` C'`)` is a reversal of q_1, and every reversal link is consistent.
$C = C_1$ `and` C_2`:` (1) $q_1 \equiv (q_0$ `and` $C_1)$ `and` C_2 (associativity), (2) there is a path from q_0 to $q_2 = (q_0$ `and` $C_1)$ by induction on C_1, (3) there is a path from q_2 to `(`q_2 `and` C_2`)` by induction on C_2. The induction in (2) is justified because q_2 is more general than q_1, and hence $ext(q_2) \neq \emptyset$.
$C =$ `not` $C_1,\ C_1 \in \mathcal{C}$`:` there is a path q_0 `[zoom-in not` C_1`]` q_1. □

In the index, the set of resources can be seen as the list of answers to the query, and can be presented page by page, like in web search engines. The set of properties is organized into a subsumption hierarchy that can be expanded on demand by users. Instead of showing up to 12 complex properties for every basic property p, only p `:` and p `of` can be displayed. Their various closures are accessible by toggling each closure on/off when applying zoom-in (see check-boxes in Figure 1). The vocabulary used to compute the index and local navigation links in our prototype is richer than the base vocabulary of Theorem 2, in order to provide richer summaries of query results. The index also contains the hierarchy of classes (`a` u), and the user can expand restrictions recursively, i.e., each P `?` is refined into $P\ C$, where C is derived in the same way the main

index is derived from `?`. Zoom-in is performed by double-clicking an index term, naming is performed by pushing a button that generates a fresh variable, and reversal is performed by clicking on a content word of the query (e.g., resource, variable, class, property). The interface offers three short-hand navigation links on index terms (semi-colon is used to compose navigation links): `[home]` (resets the query to `?`), `[pivot` C`]` = `[home; zoom-in` C`]`, `[cross` P C`]` = `[zoom-in` P C`; reversal` C`]`.

Then, the above query `a woman and parent : birth : (date : 1642 and place : in VA)` is accessible from `?` through the following navigation path: `[zoom-in a woman; cross parent : ?; cross birth : ?; zoom-in date : 1642; zoom-in place : in VA; reversal a woman]`. After selecting women, the user moves to their parents, and then to the birth of their parents. By expanding recursively index terms, the user discovers birthdates and birthplaces, and select a date (1642), and a place (in VA). Finally, the user comes back to the point of view of women, now restricted to those whose some parent was born in 1642 in Virginia.

A more complex query with a cycle and a restricted complement is: `a person and ?X and birth : date : ?D and mother : mother of (not ?X and birth : date : ?D)`, which retrieves people with a twin sibling. It can be reached through the navigation path: `[zoom-in a person; naming ?X; cross birth :; cross date :; naming ?D; reversal ?X; cross mother :; cross mother of; zoom-in not ?X; cross birth :; cross date :; zoom-in ?D; reversal ?X]`.

6 Conclusion

We have defined local views over RDF graphs that serve both for summarization and navigation. Each loval view provides a set of local links that users can follow to reach other local views. The navigation graph induced by local views is both consistent and conjunctive-complete. Compared to existing conceptual navigation systems, our query language adds variables, binary relations between objects, negation and disjunction, thus covering most of the expressivity of SPARQL. Compared to SW query languages, we provide the same benefits as faceted search, i.e., exploratory search, but a larger fragment of the query language is reachable by navigation. Furthermore, consistency and completeness of navigation are proved formally.

Preliminary results from a user evaluation shows that all subjects could answer simple questions, like "Which men were born in 1659?"; and at least half of the subjects could answer complex questions involving variables, negation or disjunction, like "Which women have a mother whose death's place is not Warner Hall?" or "Who was born the same year as his/her spouse?". The simple questions match the expressivity of other faceted search systems, while complex questions are beyond their scope.

References

[BCM+03] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[BD08] F. Baader and F. Distel. A finite basis for the set of EL-implications holding in a finite model. In R. Medina and S. A. Obiedkov, editors, *Int. Conf. Formal Concept Analysis*, LNCS 4933, pages 46–61. Springer, 2008.

[CR96] C. Carpineto and G. Romano. A lattice conceptual clustering system and its application to browsing retrieval. *Machine Learning*, 24(2):95–122, 1996.

[DE08] J. Ducrou and P. Eklund. An intelligent user interface for browsing and search MPEG-7 images using concept lattices. *Int. J. Foundations of Computer Science, World Scientific*, 19(2):359–381, 2008.

[Fer09] S. Ferré. Camelis: a logical information system to organize and browse a collection of documents. *Int. J. General Systems*, 38(4), 2009.

[FHH04] R. Fikes, P. J. Hayes, and I. Horrocks. OWL-QL - a language for deductive query answering on the semantic web. *J. Web Semantic*, 2(1):19–29, 2004.

[FR04] S. Ferré and O. Ridoux. An introduction to logical information systems. *Information Processing & Management*, 40(3):383–419, 2004.

[FRS05] S. Ferré, O. Ridoux, and B. Sigonneau. Arbitrary relations in formal concept analysis and logical information systems. In *ICCS*, LNCS 3596, pages 166–180. Springer, 2005.

[GW99] B. Ganter and R. Wille. *Formal Concept Analysis — Mathematical Foundations*. Springer, 1999.

[HHNV07] M. R. Hacene, M. Huchard, A. Napoli, and P. Valtchev. A proposal for combining formal concept analysis and description logics for mining relational data. In S. O. Kuznetsov and S. Schmidt, editors, *Int. Conf. Formal Concept Analysis*, LNCS 4390, pages 51–65. Springer, 2007.

[HKR09] P. Hitzler, M. Krötzsch, and S. Rudolph. *Foundations of Semantic Web Technologies*. Chapman & Hall/CRC, 2009.

[HvOH06] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A browser for heterogeneous semantic web repositories. In I. Cruz *et al*, editor, *Int. Semantic Web Conf.*, LNCS 4273, pages 272–285. Springer, 2006.

[Mar06] G. Marchionini. Exploratory search: from finding to understanding. *Communications of the ACM*, 49(4):41–46, 2006.

[ODD06] E. Oren, R. Delbru, and S. Decker. Extending faceted navigation to RDF data. In I. Cruz *et al*, editor, *Int. Semantic Web Conf.*, LNCS 4273, pages 559–572. Springer, 2006.

[PAG06] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. In I. F. Cruz *et al*, editor, *Int. Semantic Web Conf.*, LNCS 4273, pages 30–43. Springer, 2006.

[RKH07] S. Rudolph, M. Krötzsch, and P. Hitzler. Quo vadis, CS? - on the (non)-impact of conceptual structures on the semantic web. In U. Priss, S. Polovina, and R. Hill, editors, *Int. Conf. Conceptual Structures*, LNCS 4604, pages 464–467. Springer, 2007.

[ST09] G. M. Sacco and Y. Tzitzikas, editors. *Dynamic taxonomies and faceted search*. The information retrieval series. Springer, 2009.

[VR08] J. Völker and S. Rudolph. Lexico-logical acquisition of OWL-DL axioms. In R. Medina and S. A. Obiedkov, editors, *Int. Conf. Formal Concept Analysis*, LNCS 4933, pages 62–77. Springer, 2008.